

---

# Technical Appendix

---

Anonymous Author(s)

Affiliation

Address

email

## A Datasets

### A.1 Dataset Details

Here we summarize the datasets used in this work. Additional details on the datasets can be found in the original dataset publications.

**Zebrafish (Deisseroth).** Parts of the dataset are published in [1]. Larval zebrafish expressing nuclear-localized GCaMP6s were used for two-photon calcium imaging. Fish were head-fixed for whole-brain imaging while retaining the ability to perform tail movements. Neural activity was recorded at about 1 Hz, yielding volumetric data from 8,000–22,000 neurons per fish. The imaging data were motion-corrected, segmented into nuclei, and fluorescence traces were extracted and converted into  $\Delta F/F$ . ROIs were assigned brain region labels based on anatomical landmarks.

Experiments were conducted across four cohorts. The Spontaneous cohort was imaged during natural, unstimulated behavior. The Shocked cohort received randomly timed mild electric shocks during imaging. The Reshocked cohort was pre-exposed to a behavioral challenge before imaging, then imaged under the same shock protocol. The Ketamine cohort received ketamine exposure prior to shock delivery during imaging. There are 5 fish for each cohort, except that the Ketamine cohort has 4 fish. These conditions allowed us to examine brain-wide neural dynamics under varying levels of behavioral and pharmacological perturbation.

**Zebrafish (Ahrens).** The dataset is published in [6]. A light-sheet imaging setup is used to record whole-brain neural activity from larval zebrafish expressing nuclear-localized GCaMP6f. Imaging was performed at  $\sim 2.1$  Hz for  $\sim 50$  minutes across 18 fish, resulting in  $\sim 6,800$  time points and  $\sim 80,000$  segmented neurons per animal. Subject 8, 9, and 11 are excluded due to file incompatibility issues, thus, only 15 fish are used. During imaging, fish were exposed to a battery of visual stimuli, including phototaxis cues, moving gratings (optomotor response), looming discs (escape), dark flashes, and spontaneous illumination blocks. Although behavioral data were also collected, we used only the neural activity traces for modeling, and no stimulus or behavioral labels were provided to the model.

**Mice (Harvey).** The dataset is partly described in [13], and the mesoscopic calcium imaging technique is detailed in [2]. It includes multi-session two-photon calcium imaging recordings from layer 2/3 neurons in four cortical regions: primary visual cortex (V1), secondary motor cortex (M2), posterior parietal cortex (PPC), and retrosplenial cortex (RSC). Mice expressing GCaMP6s were head-fixed and allowed to run spontaneously on an air-supported spherical treadmill in complete darkness. Imaging was performed using a large field-of-view two-photon microscope, capturing neural activity at about 5.4 Hz from two planes per region. Data were collected across multiple sessions and animals, with each session lasting 45–60 minutes. Four mice were recorded, with 1, 5, 4, and 2 sessions respectively. For the first two mice, all four regions were recorded. For the third mouse, two sessions included simultaneous recordings of PPC, RSC, and V1, while the other two sessions included PPC, RSC, and M2. For the fourth mouse, only PPC, RSC, and V1 were recorded. Motion correction and ROI extraction were performed using standard pipelines. Note that no behavioral or anatomical labels were provided to the model.

40 ***C. elegans* (Zimmer).** The dataset is described in [8]. This dataset contains whole-brain calcium  
 41 imaging recordings from *C. elegans*, using a pan-neuronally expressed nuclear calcium indicator  
 42 GCaMP5K. Neural activity was recorded from 5 animals at 2.85 volumes per second for 18 minutes per  
 43 animal. The imaging volume covered the head ganglia, including most sensory neurons, interneurons,  
 44 all head motor neurons, and the anterior ventral cord motor neurons. Each recording included  
 45 107–131 neurons, with most active neurons identifiable by cell class. Animals were recorded either  
 46 under constant oxygen or during alternating oxygen stimuli. Only neural activity traces were used in  
 47 our experiments; stimulus and cell identity labels were not provided to the model.

48 ***C. elegans* (Flavell).** The dataset is published in [3]. This dataset includes brain-wide calcium imaging  
 49 recordings from freely moving *C. elegans* using a strain expressing nuclear-localized GCaMP7f and  
 50 mNeptune2.5 in all neurons. Recordings were conducted on a custom dual-light-path microscope  
 51 with closed-loop tracking to maintain the worm’s head within the imaging field. Neural signals were  
 52 extracted using automated segmentation and tracking based on 3D U-Net and registration pipelines.  
 53 We used only the 40 sessions with NeuroPAL neuron identity labels—21 recorded under baseline  
 54 conditions and 19 with a noxious heat stimulus that induced a persistent behavioral state change.  
 55 Only the neural activity traces were used in our experiments; stimulus timing and neuron identity  
 56 labels were not provided to the model.

57 **Zapbench.** The benchmark is described in [10]. This dataset consists of whole-brain calcium  
 58 imaging from a single head-fixed larval zebrafish recorded during fictive behavior in a virtual reality  
 59 environment. Over the course of a two-hour session, the fish was exposed to nine structured visual  
 60 stimulus conditions designed to elicit a range of sensorimotor responses. Neural activity was recorded  
 61 at cellular resolution using light-sheet fluorescence microscopy, resulting in a 4D volumetric dataset  
 62 spanning 71,721 segmented neurons. Postprocessing included motion correction, custom elastic  
 63 alignment, and manual neuron segmentation using a Flood-Filling Network. Only the extracted  
 64 per-neuron calcium traces were used in our experiments; stimulus information was not provided to  
 65 the model.

## 66 A.2 Data Processing

67 For Zapbench, we follow the code provided with the benchmark to directly load the processed data  
 68 [10]. For other datasets, we begin by z-scoring the raw fluorescence values for each neuron so that  
 69 each calcium trace has zero mean and unit variance. This normalization ensures that all neurons are  
 70 equally weighted in the model—an active neuron with a high baseline activity contributes similarly  
 71 to a less active one. For Deisseroth’s zebrafish dataset, where raw calcium traces are unavailable,  
 72 we instead apply z-scoring to the denoised activity extracted using constrained nonnegative matrix  
 73 factorization (CNMF) [14]. For experiments involving frequency filtering, we apply a zero-phase  
 74 fourth-order Butterworth filter. The default low-pass filter removes frequency components above  
 75  $0.1 \times f_s$ , where  $f_s$  is the sampling frequency. A band-pass filter additionally removes components  
 76 below  $5 \times 10^{-3} \times f_s$ . After filtering, we optionally reduce dimensionality using principal component  
 77 analysis (PCA), applied to the z-scored (and optionally filtered) neural activity matrix. The magnitudes  
 78 of the resulting principal components are preserved, meaning that dominant components naturally  
 79 carry more weight in the loss function.

## 80 A.3 Data Partitioning

81 We first divide each session into segments of 1,000 time steps. Each segment is then split into training,  
 82 validation, and test partitions using a 3:1:1 ratio. The final segment of a session may contain more  
 83 than 1,000 steps to maximize use of the available data. To construct the training set, we extract  
 84 all possible consecutive subsequences of length  $C + P$  (64 by default) from the training partition  
 85 using a sliding window with stride 1. For example, if a session contains 2,500 steps, the training  
 86 partitions would cover steps [1, 600] and [1001, 1900], and the training subsequences would include  
 87  $\mathbf{x}_{1:1+C+P}, \mathbf{x}_{2:2+C+P}, \dots, \mathbf{x}_{600-C-P+1:600}$  and similarly for the second partition. Validation and  
 88 test sets are constructed in the same way. However, because evaluation is computationally expensive  
 89 for single-cell resolution zebrafish datasets, we use a larger stride when extracting subsequences  
 90 from the validation set: stride 32 for Ahrens’ dataset and stride 8 for Deisseroth’s dataset. Note that  
 91 striding is applied only to the validation set in the context of single-cell level prediction.

For Zapbench, we instead use the provided tools to load the dataset [10]. One notable difference is that in Zapbench, the dataset is first divided by the stimulus condition then divide each condition into train, validation and test sets.

## B Additional Model Details

### B.1 POCO

In POCO, we use Rotary Position Embeddings (RoPE) in all attention layers [15]. In RoPE, a timestamp  $t$  is assigned to each query token  $\mathbf{q}_i$  and key token  $\mathbf{k}_j$ , and the attention score is computed as

$$\mathbf{a}_{ij} = \text{softmax} \left( (\mathbf{R}(t_i) \mathbf{q}_i)^T (\mathbf{R}(t_j) \mathbf{k}_j) \right), \quad (1)$$

where  $\mathbf{R}(t)$  is a time-dependent rotation matrix. We follow POYO [4] to construct  $\mathbf{R}(t)$ , defined as a composition of  $2 \times 2$  rotation matrices with periods  $T_i$  logarithmically spaced between  $T_{\min}$  and  $T_{\max}$ . In our experiments, we use  $T_{\min} = 10^{-3} \times T_{\max}$  and set  $T_{\max} = 100$  by default.

We also need to define the timestamps. For the first attention layer,

$$\mathbf{L}_1 = \text{Attention}_0(Q = \mathbf{L}_0; K, V = \mathbf{E}) \in \mathbb{R}^{N_L \times d}, \quad (2)$$

we assign timestamps to the trace tokens  $\mathbf{E}(i, k)$  as  $t_{i,k} = (k - 1)T_C$ , which corresponds to the starting timestep of the token. For the latent tokens  $\mathbf{L}_0(j)$ , we set

$$t_j = \frac{j - 1}{N_L} C, \quad j \in [N_L], \quad (3)$$

so that the timestamps uniformly span the input context length  $C$ . Timestamp for latents in later attention layers  $\mathbf{L}_l$  are defined in the same way. For the final layer, the query tokens are given a constant time step  $t = C$ . Intuitively, the latents can be viewed as encoding the global population state at different moments in the past, and in the final layer, we query how those past states influence the future.

For Zapbench experiments, we modify these parameters to accommodate shorter or longer context lengths: specifically, we use  $T_{\max} = 50$ ,  $T_C = 4$  for  $C = 4$ , and  $T_{\max} = 400$ ,  $T_C = 64$  for  $C = 256$ . The default token length is  $T_C = 16$ , resulting in 3 tokens per neuron when  $C = 48$ .

For the conditioning layer, we initialize the weights  $\mathbf{W}_\beta$ ,  $\mathbf{W}_\gamma$  and biases  $\mathbf{b}_\beta$ ,  $\mathbf{b}_\gamma$  of the linear mappings (from the final attention layer output to FiLM conditioning parameters) to zero, ensuring that POCO produces a flat output at initialization. We follow POYO on additional model details: We use an additional feedforward layer following each attention layer. Residual connections are used for both the attention layers and the feedforward layers that follow. We use 1 attention head for the first and last attention layer, and 8 heads for intermediate self-attention layers. We set the embedding and latent size to  $d = 128$ ; in rotary attention, each dimension of each head is 64. We use a dropout of 0.2 in the feed-forward layers following the attention layers and a dropout of 0.4 on the output of intermediate self-attention layers and the accompanying feedforward layers. We initialize the embeddings from  $\mathcal{N}(0, \sigma_0^2)$  with  $\sigma_0 = 0.02$ .

### B.2 Ablation Study

In the ablation study, we considered several alternative architectures. First, to directly use the POYO model to generate 16-steps prediction, we use a linear projection on top of the output of the last layer

$$\begin{aligned} \mathbf{L}_{L+2} &= \text{Attention}_{L+1}(Q = \mathbf{U}_j; K, V = \mathbf{L}_{L+1}) \in \mathbb{R}^{N_j \times d}, \\ \hat{\mathbf{x}}_{t:t+P} &= \mathbf{W}_{out} \mathbf{L}_{L+2}^T + \mathbf{b}_{out}, \end{aligned} \quad (4)$$

where  $\mathbf{W}_{out} \in \mathbb{R}^{P \times d}$ .

Another alternative we tested is replacing the population encoder with a univariate Transformer, which takes the calcium trace of a single neuron instead of encoding the whole population. Specifically, for each neuron  $i$ , we still use  $T_C = 16$  steps to form one token, but the token embedding no longer involves the unit and session embedding:

$$E(k) = \mathbf{W} x_{r_k - T_C:r_k, i}^{(j)} + \mathbf{b}. \quad (5)$$

132 We then apply a standard 1-layer transformer encoder used in recent time-series forecasting work  
 133 [16]. Note that the Transformer is applied separately to each neuron. We concatenate hidden states of  
 134 the last encoder layer, resulting in a  $dC/T_C = 128 \times 3$ -dimensional vector for each neuron. Finally,  
 135 we generate 16-step prediction by applying a linear projection on the concatenated hidden state.

### 136 B.3 Other Baselines

137 **NLinear.** In NLinear [19], the last step of the input context is subtracted from the input sequence,  
 138 and the residual is passed through a linear layer to generate predictions ( $\mathbb{R}^C \rightarrow \mathbb{R}^P$ ). The last step is  
 139 then added back to the output. This design is effective in time series forecasting (TSF) tasks as it  
 140 helps address distributional shifts between training and test sets. NLinear is a univariate model—each  
 141 neuron’s prediction depends only on its own history—which also makes it naturally capable of  
 142 processing multi-session data, as it can handle an arbitrary number of neurons.

143 **DLinear.** We adopt the implementation from [19]. DLinear first decomposes each time series into a  
 144 trend and a seasonal component. The trend component is obtained by applying a moving average filter  
 145 with kernel size  $2\lfloor C/4 \rfloor + 1$ , and the seasonal component is the residual obtained by subtracting the  
 146 trend from the original input. Both components are predicted independently using linear projections,  
 147 and the final output is their sum. Like NLinear, DLinear is also univariate.

148 **MLP.** We use a two-layer MLP with ReLU activation and hidden size 1024. This model is equivalent  
 149 to POCO without FiLM conditioning. It is also univariate.

150 **Latent PLRNN.** We include PLRNN as a baseline due to its demonstrated ability to reconstruct  
 151 chaotic dynamical systems. We use the implementation from [5]. The model defines a dynamical  
 152 system over a  $d$ -dimensional latent state vector  $\mathbf{z}_t$  as:

$$\mathbf{z}_t = \mathbf{A}\mathbf{z}_{t-1} + \mathbf{W}\phi(\mathbf{z}_{t-1}) + \mathbf{h},$$

153 where  $\mathbf{A} \in \mathbb{R}^{d \times d}$  is a diagonal matrix encoding self-connections,  $\mathbf{W} \in \mathbb{R}^{d \times d}$  contains off-diagonal  
 154 weights for inter-unit interactions, and  $\mathbf{h} \in \mathbb{R}^d$  is a bias term. The activation function  $\phi$  is ReLU. We  
 155 set  $d = 512$ . To generate predictions, we first map the last step from the context to the latent state:

$$\mathbf{z}_C = \mathbf{W}_{\text{in}}\mathbf{x}_C + \mathbf{b}_{\text{in}},$$

156 then evolve the latent state for  $P = 16$  steps and map it back to the observation space:

$$\hat{\mathbf{x}}_{C+t} = \mathbf{W}_{\text{in}}^\dagger(\mathbf{z}_{C+t} - \mathbf{b}_{\text{in}}), \quad t \in [P],$$

157 where  $\mathbf{W}_{\text{in}}^\dagger$  denotes the pseudo-inverse of  $\mathbf{W}_{\text{in}}$ . PLRNN is multivariate, but it only uses the last  
 158 time step of the context to generate prediction. To fully utilize the input during training, we instead  
 159 initialize the latent state  $\mathbf{z}_1$  from  $\mathbf{x}_1$  and evolve the latent dynamics model for  $C + P - 1$  steps. We  
 160 apply sparse teacher forcing (STF) [12], injecting the ground-truth state every four steps to reduce the  
 161 drift from the groundtruth for effective training; STF is disabled during the final  $P$  steps to simulate  
 162 free prediction. For multi-session training, the latent dynamics (PLRNN parameters) are shared  
 163 across sessions, while the input mapping  $\mathbf{W}_{\text{in}}$  is session-specific to capture individual variability.

164 **AR\_Transformer.** We also include an autoregressive Transformer [17] as a baseline to assess the  
 165 performance of classical autoregressive architectures in neural population modeling. A linear layer  
 166 ( $\mathbb{R}^N \rightarrow \mathbb{R}^d$ ) maps each time step’s population activity to a  $d$ -dimensional embedding, producing  $C$   
 167 tokens. Here we use  $d = 512$ . These tokens are passed through a 4-layer Transformer with causal  
 168 attention masks. A final linear projection ( $\mathbb{R}^d \rightarrow \mathbb{R}^N$ ) maps the output embeddings to next-step  
 169 predictions. To forecast  $P = 16$  steps, the model is applied autoregressively: at each step, the  
 170 predicted activity is appended and used as input for the next prediction. As with PLRNN, for multi-  
 171 session training, the Transformer backbone is shared, while the input and output projection layers are  
 172 session-specific.

173 **TSMixer.** TSMixer [7] is an all-MLP architecture for time series forecasting. It consists of stacked  
 174 mixer blocks, each comprising a time mixer (a linear projection operating along the time dimension)  
 175 and a feature mixer (an MLP with hidden size 64 that operates across the neuron dimension). The in-  
 176 clusion of the feature mixer makes TSMixer a multivariate model. We use the default hyperparameter  
 177 settings: two mixer blocks and a dropout rate of 0.1.

178 **TexFilter.** TexFilter [18] is a recent MLP-based method that incorporates frequency-domain filtering.  
 179 It applies a context-dependent filter on the Fourier-transformed input, allowing the model to selectively

enhance or suppress specific frequency components. In addition to the MLP layer, TexFilter uses learnable complex embeddings that modulate the frequency representation. TexFilter is univariate. In our setup, we set the embedding size to 128 and initialize the embeddings from  $\mathcal{N}(0, \sigma_0^2)$ ,  $\sigma_0 = 0.02$ , with a dropout rate of 0.3.

**TCN.** ModernTCN [11] employs modernized convolutional blocks for time series modeling. The input is first divided into patches of size  $P = 4$ , which are embedded into  $D = 64$ -dimensional vectors. These embeddings are processed using convolutional blocks that capture both temporal and cross-neuron dependencies, followed by a linear readout for prediction. This makes ModernTCN a multivariate model. We follow the original work’s hyperparameters for forecasting, except for the convolutional kernel sizes, which are adjusted to 17 (large) and 5 (small) to better suit our shorter context window.

**Netformer.** Netformer [9] is a recent approach for modeling dynamical connectivity in neural population activity. It embeds each neuron’s past activity trace and uses an attention layer to compute interaction weights  $A$  across neurons. The next-step prediction is given by:

$$\hat{\mathbf{x}}_{t+1} = A\mathbf{x}_t + \mathbf{x}_t.$$

Although the original work only evaluated next-step prediction, we extend it to multi-step forecasting by recursively applying the same update. However, we observed instability when training on long horizons and therefore applied a softmax to the attention weights to stabilize learning. Furthermore, to further improve stability, we also applied an instance normalization layer on the input and later unnormalized the prediction as in recent TSF works [18, 16]. As in the original Netformer, we apply layer normalization along the time dimension, and use an embedding size of 30. Netformer is a multivariate model but is naturally compatible with multi-session training, as the attention mechanism is independent of the number of neurons.

## C Experiments

### C.1 Training

For multi-session training, we train each model for  $10^4$  steps. Single-session models are instead trained for  $5 \times 10^3$  steps considering the smaller training set size. At each training step, a batch of 64 sequences is sampled from the training sets of all sessions. For single-cell prediction in larval zebrafish, we reduce the batch size to 8 for Deisseroth’s zebrafish dataset and to 4 for Ahrens’ zebrafish dataset due to memory constraints. Gradient clipping is applied to limit the gradient norm to 5. Most models are trained using mean squared error (MSE) loss averaged over the  $P$  prediction steps. However, for PLRNN, the autoregressive Transformer, and Netformer—models that generate one-step predictions—the loss is computed across all  $C + P - 1$  steps. Models are evaluated on the validation set every 100 training steps, and we report test-set performance using the checkpoint with the lowest validation loss.

### C.2 Simulation

We generate synthetic neural data from a chaotic RNN with a tanh nonlinearity, governed by the following dynamical equation:

$$\mathbf{r} = \tanh(\mathbf{h}), \quad \tau \frac{d\mathbf{h}}{dt} = -\mathbf{h} + g\mathbf{J}\mathbf{r} + \sqrt{2\tau\sigma^2}\boldsymbol{\xi}, \quad \mathbf{h}(0) \sim \mathcal{U}[-1, 1],$$

where  $\mathbf{h}$  is the internal (pre-activation) state,  $\mathbf{r} = \tanh(\mathbf{h})$  is the firing rate,  $\mathbf{J}$  is a recurrent weight matrix,  $g = 2.0$  is a gain factor,  $\boldsymbol{\xi}$  are  $N$  independent Gaussian white noise processes with zero mean and unit variance,  $\sigma = 0.1$  controls the noise variance, and  $\tau = 0.1$  is the time constant. We discretize the dynamics using Euler’s method with time step  $\Delta t = 0.01$ :

$$\mathbf{h}_{t+1} = \mathbf{h}_t + \frac{\Delta t}{\tau} (-\mathbf{h}_t + g\mathbf{J}\tanh(\mathbf{h}_t)) + \sqrt{2\sigma^2 \frac{\Delta t}{\tau}} \boldsymbol{\xi}_t, \quad \mathbf{r}_t = \tanh(\mathbf{h}_t),$$

where  $\boldsymbol{\xi}_t \sim \mathcal{N}(0, I)$  at each step. We z-score the firing rates  $\mathbf{r}_t$ , and to simulate the slower sampling rate of calcium imaging, we average  $\mathbf{r}_t$  over every  $f = 5$  time steps. The simulation runs from  $t = 0$  to  $t = 4096 \times f\tau$ , producing 4096 time steps of synthetic data—comparable in length to real neural recordings.

### 225 C.3 Finetuning

226 For pretraining and finetuning experiments, we partition each dataset and use approximately 80%  
 227 of the sessions for pretraining. Specifically, for Deisseroth’s zebrafish dataset, we reserve the last  
 228 session from each cohort for finetuning, yielding 15 sessions for pretraining. For Ahrens’ zebrafish  
 229 dataset, we finetune on the last 4 sessions and pretrain on the remaining 11. For the mice dataset  
 230 from Harvey et al., we finetune using the 2 sessions from the last mouse. All models are finetuned for  
 231 2,000 steps and evaluated every 20 steps.

### 232 C.4 Unit Embedding

233 Although multi-session POCO learns a shared unit embedding space across sessions, the additional  
 234 session embedding may introduce session-specific offsets. To avoid this confound, we analyze and  
 235 visualize unit embeddings separately for each session. Let  $U(i)$  denote the embedding of neuron  $i$ ,  
 236 and  $S_u^{(j)}$  the set of neurons in region  $u$  from session  $j$ . The cosine similarity between regions  $u$  and  $v$   
 237 is computed as:

$$C(u, v) = \frac{1}{\sum_j |S_u^{(j)}| |S_v^{(j)}|} \sum_j \sum_{i_u \in S_u^{(j)}} \sum_{i_v \in S_v^{(j)}} \frac{U(i_u)^T U(i_v)}{\|U(i_u)\| \|U(i_v)\|}, \quad \text{for } u \neq v,$$

$$C(u, u) = \frac{1}{\sum_j |S_u^{(j)}| (|S_u^{(j)}| - 1)} \sum_j \sum_{\substack{i_u, i_v \in S_u^{(j)} \\ i_u \neq i_v}} \frac{U(i_u)^T U(i_v)}{\|U(i_u)\| \|U(i_v)\|}.$$

238 To highlight relative similarity patterns, we compute a row-wise normalized similarity:

$$\bar{C}(u, v) = \frac{C(u, v) - \min_{v'} C(u, v')}{\max_{v'} C(u, v') - \min_{v'} C(u, v')}.$$

239 This normalized similarity more clearly reveals which brain regions are functionally closer or further  
 240 from a given region  $u$ . The same analysis can be applied using other distance metrics, such as  
 241 Euclidean distance.

### 242 C.5 Multi-Dataset Training

243 To support multi-dataset training, we assign a separate dataloader to each dataset. At each training  
 244 step, we sample one batch from each dataset, compute the loss independently, and sum the losses  
 245 before performing a single backward pass. To help POCO handle cross-dataset differences, we add a  
 246 dataset-specific embedding to each input token:

$$E(i, k) = \mathbf{W} x_{r_k - T_C:r_k, i}^{(j)} + \mathbf{b} + \text{UnitEmbed}(i, j, u) + \text{SessionEmbed}(j, u) + \text{DatasetEmbed}(u),$$

247 where  $u \in [D]$  is the dataset index and  $D$  is the total number of datasets.

### 248 C.6 Experiments on Zapbench

249 For Zapbench, we follow the provided script for dataloading and testing [10]. We train POCO for 25  
 250 epochs with a batch size of 8, validating after every epoch. We report test-set performance using the  
 251 checkpoint with the lowest validation loss. Following the benchmark protocol, we use mean absolute  
 252 error (MAE) as the loss function. The *TAXIS* condition is excluded during training, and performance  
 253 is evaluated on the remaining stimulus conditions.

## 254 D Compute Resources

255 All models are trained on NVIDIA A100 or H100 GPUs paired with AMD EPYC 9454 CPUs. Only  
 256 a single CPU thread is used throughout training. Data preprocessing for all datasets takes less than  
 257 one CPU day in total, and generating all synthetic data requires under two CPU days.

258 For multi-session prediction of the first 512 principal components (PCs) from Deisseroth’s zebrafish  
 259 dataset, training POCO for  $10^4$  steps takes less than 30 minutes, including data loading and validation

time. Despite using a high-end GPU, this training setup consumes under 4GB of GPU memory. Computational cost increases for datasets with more neurons: for single-cell prediction on Ahrens’ zebrafish dataset, training consumes up to 48GB of GPU memory and completes in under one hour.

While training a single model is fast, some experiments involve training many models. For example, training a single-session model for each session in each dataset across 4 random seeds involves training over 300 models in total. The estimated total GPU time across all experiments is approximately 1,500 hours.

The same hardware setup (H100 GPU) is used to measure finetuning and inference speed. When finetuning POCO on the first 512 PCs of Deisseroth’s dataset, we find that adapting the embedding for 200 training steps takes less than 15 seconds, and forecasting a single sequence takes only 3.5 milliseconds.

## References

- [1] A. S. Andalman, V. M. Burns, M. Lovett-Barron, M. Broxton, B. Poole, S. J. Yang, L. Grose, T. N. Lerner, R. Chen, T. Benster, et al. Neuronal dynamics regulating brain and behavioral state transitions. *Cell*, 177(4):970–985, 2019.
- [2] C. Arlt, R. Barroso-Luque, S. Kira, C. A. Bruno, N. Xia, S. N. Chettih, S. Soares, N. L. Pettit, and C. D. Harvey. Cognitive experience alters cortical involvement in goal-directed navigation. *Elife*, 11:e76051, 2022.
- [3] A. A. Atanas, J. Kim, Z. Wang, E. Bueno, M. Becker, D. Kang, J. Park, T. S. Kramer, F. K. Wan, S. Baskoylu, et al. Brain-wide representations of behavior spanning multiple timescales and states in *c. elegans*. *Cell*, 186(19):4134–4151, 2023.
- [4] M. Azabou, V. Arora, V. Ganesh, X. Mao, S. Nachimuthu, M. Mendelson, B. Richards, M. Perich, G. Lajoie, and E. Dyer. A unified, scalable framework for neural population decoding. *Advances in Neural Information Processing Systems*, 36:44937–44956, 2023.
- [5] M. Brenner, F. Hess, J. M. Mikhaeil, L. F. Bereska, Z. Monfared, P.-C. Kuo, and D. Durstewitz. Tractable dendritic rnns for reconstructing nonlinear dynamical systems. In *International conference on machine learning*, pages 2292–2320. Pmlr, 2022.
- [6] X. Chen, Y. Mu, Y. Hu, A. T. Kuan, M. Nikitchenko, O. Randlett, A. B. Chen, J. P. Gavornik, H. Sompolinsky, F. Engert, et al. Brain-wide organization of neuronal activity and convergent sensorimotor transformations in larval zebrafish. *Neuron*, 100(4):876–890, 2018.
- [7] V. Ekambaram, A. Jati, N. Nguyen, P. Sinthong, and J. Kalagnanam. Tsmixer: Lightweight mlp-mixer model for multivariate time series forecasting. In *Proceedings of the 29th ACM SIGKDD conference on knowledge discovery and data mining*, pages 459–469, 2023.
- [8] S. Kato, H. S. Kaplan, T. Schrödel, S. Skora, T. H. Lindsay, E. Yemini, S. Lockery, and M. Zimmer. Global brain dynamics embed the motor command sequence of *caenorhabditis elegans*. *Cell*, 163(3):656–669, 2015.
- [9] Z. Lu, W. Zhang, T. Le, H. Wang, U. Sümbül, E. T. SheaBrown, and L. Mi. Netformer: An interpretable model for recovering dynamical connectivity in neuronal population dynamics. In *The Thirteenth International Conference on Learning Representations*, 2025. URL <https://openreview.net/forum?id=bcTjW5kS4W>.
- [10] J.-M. Lueckmann, A. Immer, A. B.-Y. Chen, P. H. Li, M. D. Petkova, N. A. Iyer, L. W. Hesselink, A. Dev, G. Ihrke, W. Park, et al. Zapbench: A benchmark for whole-brain activity prediction in zebrafish. *arXiv preprint arXiv:2503.02618*, 2025.
- [11] D. Luo and X. Wang. Modernctcn: A modern pure convolution structure for general time series analysis. In *The twelfth international conference on learning representations*, pages 1–43, 2024.
- [12] J. Mikhaeil, Z. Monfared, and D. Durstewitz. On the difficulty of learning chaotic dynamics with rnns. *Advances in neural information processing systems*, 35:11297–11312, 2022.

- 307 [13] M. G. Perich, C. Arlt, S. Soares, M. E. Young, C. P. Mosher, J. Minxha, E. Carter, U. Rutishauser,  
308 P. H. Rudebeck, C. D. Harvey, et al. Inferring brain-wide interactions using data-constrained  
309 recurrent neural network models. *BioRxiv*, pages 2020–12, 2020.
- 310 [14] E. A. Pnevmatikakis, D. Soudry, Y. Gao, T. A. Machado, J. Merel, D. Pfau, T. Reardon, Y. Mu,  
311 C. Lacefield, W. Yang, et al. Simultaneous denoising, deconvolution, and demixing of calcium  
312 imaging data. *Neuron*, 89(2):285–299, 2016.
- 313 [15] J. Su, M. Ahmed, Y. Lu, S. Pan, W. Bo, and Y. Liu. Roformer: Enhanced transformer with  
314 rotary position embedding. *Neurocomputing*, 568:127063, 2024.
- 315 [16] S. Talukder, Y. Yue, and G. Gkioxari. Totem: Tokenized time series embeddings for general  
316 time series analysis. *arXiv preprint arXiv:2402.16412*, 2024.
- 317 [17] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and  
318 I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*,  
319 30, 2017.
- 320 [18] K. Yi, J. Fei, Q. Zhang, H. He, S. Hao, D. Lian, and W. Fan. Filtarnet: Harnessing frequency  
321 filters for time series forecasting. *Advances in Neural Information Processing Systems*, 37:  
322 55115–55140, 2024.
- 323 [19] A. Zeng, M. Chen, L. Zhang, and Q. Xu. Are transformers effective for time series forecasting?  
324 In *Proceedings of the AAAI conference on artificial intelligence*, volume 37, pages 11121–11128,  
325 2023.